WEST

□ Generate Collection

L12: Entry 22 of 25 File: USPT Jan 30, 1996

DOCUMENT-IDENTIFIER: US 5488610 A
TITLE: Communication system

Detailed Description Paragraph Right (13):
At the remote device the telephone will ring. Its modem will send a RING response message to the communications card of the receiving mux 22a which sets a status to indicate that this device is the answering station. The other user lifts the telephone handset and a normal voice conversation can ensue.

Detailed Description Paragraph Right (16):
When the modems 26 and 26a have trained, a CONNECT response is sent to the communications cards of the muxes 22 and 22a which then execute the multiplexer code. When the multiplexers have synchronised, a message is displayed at the screens 28 and 28a and the users may now proceed with a voice and data call.

Detailed Description Paragraph Right (28):
The application layer 44 takes pen input from the digitising tablet and codes it into sketching information. It also displays images and sketching on the display. Via the lower layers of the communication stack it communicates changes at the pen and screen interface to a remote voice and data device. In the embodiments described, the application layer communicates with the protocol layer using software messages.

Detailed Description Paragraph Right (31):
The structure of the multiplexer frames is optimised according to `long term` requirements of the application and protocol layers, e.g. voice coder data rate and frame rate, data bandwidth requirements . . . It may change during the call (after negotiation by the protocol layers) and it need not be the same in the transmit and receive directions. The multiplexer places a number of constraints on the frame structure which ensure that clock error 1, as defined above, is eliminated. These constraints will be defined in the descriptions of the two embodiments of the multiplexer which follow.

Detailed Description Paragraph Right (32):
FIG. 5a-5g shows the structures of some possible mux frames. The length of a frame is fixed by the protocol being used at the relevant time, although the frame length can be altered from time to time to optimise operation according to which applications are active. The type of frame transmitted changes according to the `short term` requirements of the application and protocol layers, e.g. what data must be transmitted at a particular point in time. The various fields in the mux frames will now be explained.

WEST

□ Generate Collection

L12: Entry 19 of 25 File: USPT Mar 4, 1997

DOCUMENT-IDENTIFIER: US 5608720 A
TITLE: Control system and operations system interface for a network element in an access system

Abstract Paragraph Left (1):
A control system for a network element (NE) such as a remote digital terminal (RDT) in an integrated digital loop carrier (DLC) system is provided which comprises subsystems for operating circuit packs and for providing functions that are common to the circuit packs. Each subsystem has a subagent residing on a network processor circuit pack. The subagents communicate via a common subagent interface. A NE which supports both OSI and non-OSI interfaces such as Transaction Language 1 (TL1) and Common Management Services Element (CMISE) interfaces is provided, along with a method for mapping TL1 commands to CMISE messages using TL1 proxy objects.

Brief Summary Paragraph Right (1):
A network element (NE) in an integrated digital loop carrier or other access system system is provided which operates in accordance with a control system comprising subsystems in software for operating circuit packs in the system. The control system provides agents and subagents for each subsystem. Communication between the subsystems occurs via common messaging on a subagent interface (SAI). A NE which supports both Transaction Language 1 (TL1) and Common Management Services Element (CMISE) interfaces is also provided, along with a method for mapping TL1 messages to CMISE messages.

Brief Summary Paragraph Right (9):
In the non-OSI, e.g., TL1 architecture, there are only three layers (i.e., the Network, Datalink, and Physical layers), as shown in FIG. 2A. The operations application messages are mapped directly to the network layer or datalink layer of the OSI reference model. The application messages for operations in the non-OSI environment are defined using TL1. The RDTs based on the non-OSI architecture support at least one of the following two protocols: X.25 Permanent Virtual Circuit (PVC) or X.25 Switched Virtual Circuit (SVC). As an example, the following TL1 command is used to edit the information associated with a particular T1 line:

Brief Summary Paragraph Right (11):
By contrast, in the OSI architecture, there are more layers and operations application messages mapped to layer 7 of the OSI reference model. The operation messages are written using Common Management Information Services (CMIS), and the Remote Operations Service (ROS) is used to support CMIS. The particular application-service-elements that provide CMIS and ROS services are referred to as CMISE and ROSE, respectively. The transfer syntax for the messager is Abstract Syntax Notation 1 (ASN.1). FIG. 2B shows the protocol stack for the NGDLC EOC. As shown in the figure, layers 3, 4 and 5 of the OSI reference model are not used, but a convergence function is provided.

Brief Summary Paragraph Right (24):
The TL1-only implementation handles the above command in the same way as it did the previous command, except that it will modify two fields instead of one field in the appropriate MIB array element. The CMISE-only implementation uses two CMIS M-SET service messages to get the equivalent effect. The reason is that the "ds1FrameFormat" attribute belongs to another object class (other than "DS1Line Termination" object class), namely the "DS1 Framed Path Termination" object class.

Both the M-SET service messages are treated independently.

Brief Summary Paragraph Right (28):
In accordance with yet another aspect of the present invention, the subagent interface (SAI) interfaces to each of the subsystems through a name resolution function to route messages to a particular task using a subagent object instance identifier (SAOID) for which the message is targeted. By allowing applications to direct messages toward objects rather than specific tasks, the details about which subsystem is responsible for handling a particular message and the internal organization of subsystems is concealed.

Drawing Description Paragraph Right (14):
FIG. 12 illustrates the fields in an Internal Communication Network (ICN) message in accordance with the present invention;

Drawing Description Paragraph Right (15):
FIG. 13 illustrates an ICN message as viewed by the ICN;

Drawing Description Paragraph Right (16):
FIG. 14 illustrates the application portion of an ICN message which is processed by an application;

Drawing Description Paragraph Right (17):
FIG. 15 illustrates the subfields of the inter-process message header (IMH) field of an ICN message;

Drawing Description Paragraph Right (18):
FIG. 16 is a table of states for the message flag field in the IMH of an ICN message;

Drawing Description Paragraph Right (33):
FIGS. 40, 41, 42 and 43 are TL1 agent tables used to map a TL1 command into a subagent interface message in accordance with the present invention;

Detailed Description Paragraph Right (2):
Integrated Digital Loop Carrier System Generic Requirements, Objectives, and Interface: Operations (ROS/CMIS/ASN.1) Messages Release 1.0 Bellcore Technical Reference TR-TSY-000303, Revision 1, August 1992

Detailed Description Paragraph Right (42):
The Subagent Interface (SAI) 400 provides a virtual machine interface for the application agents and subagents on the NEP 62. This interface is an object-oriented message interface. Agents and subagents communicate with each other by sending messages to objects. The interface is defined in terms of the objects that are present in the interface and the services they provide. The invoker of a service need not know explicitly which subagent is providing a particular service in order to use it. All of the services provided by one object are preferably provided by one subagent.

Detailed Description Paragraph Right (48):
The communication protocol architecture for the RDT application software comprises a layered protocol stack that is used for inter-application and intra-application message communication. This stack can be approximately modeled on the OSI seven layer model, with the application layer converging onto the transport layer. The lowest four layers (FIG. 6) are implemented by the System Services Subsystem 416 (FIG. 5). These layers provide delivery of variable sized messages, handling of redundant physical circuit packs and communication media, routing to logical applications, a reliable delivery option, a non-guaranteed delivery option, and circuit pack level flow control.

Detailed Description Paragraph Right (49):
The application layer contains an Inter-Process Message Header (IMH 446 in FIG. 14) that is used by applications to manage multiple transactions with other applications. This is preferably required for all application software messages. Session establishment occurs between circuit packs in the RDT. Applications on a

physical circuit pack are considered to have a session established with an originating application if the physical circuit pack supporting the destination application has a session established with the physical circuit pack supporting the originating application. The RDT application layer provides the following functions: indicates whether a message is solicited or unsolicited; indicates that a message is a command, response or event; provides a means for requesting an acknowledgement to a message; provides a means to acknowledge a message by posting the original message back to the originator; provides a means for indicating and detecting retransmissions; and provides a means for indicating message delivery errors occurring above the transport layer (FIG. 6).

Detailed Description Paragraph Right (50):
The application protocol layer (FIG. 6) is defined by each application. Additional protocol headers (448 in FIG. 14) for the application layer is subsystem dependent. Each application provides the following functionality: I/O timing and retransmission; error recovery timing and command/response and event/acknowledgement pairing; and application flow control (which is optional). All application level inter-process communication preferably uses this protocol stack. Messages generally consist of the following components: ICN Header 444, followed by an Inter-Process Message Header 446, an optional Application Header 448, and finally an optional Message Payload 450, as shown in FIG. 12. Both the ICN and Inter-Process Message headers are of fixed size across the RDT software. The arrow at the left in FIG. 12 indicates the beginning of the application data area in a message. This is the demarcation between elements of the message that are managed by the ICN, and elements of the message that are managed by the application. The Inter-Process Message Header (IMH) starts in the first byte after the ICN header. The application header and/or message payload, if present, start in the first byte after the IMH.

Detailed Description Paragraph Right (51):
The lower layers (FIG. 6) of the communication stack are implemented by the System Services Subsystem 416. This subsystem provides a component called the Internal Communication Network (ICN) 486 (FIG. 22). The ICN provides the delivery mechanism for applications within the RDT. This delivery mechanism provides functionality associated with the lower four layers of the OSI model protocol stack shown in FIG. 6. This delivery mechanism applies to intra-circuit pack messages and inter-circuit pack messages. The ICN supports reliable message delivery between communicating applications, while decoupling physical board addresses and task identifiers from the addresses known to the applications. A specific application task is addressed with its Module Identifier (MID) and Application Class Identifier (ACID). Module Identifiers are assigned system wide, and each MID represents a circuit pack in the RDT. Circuit packs that are not on the internal network are not assigned MIDs. ACIDs are preferably assigned for all of the application tasks on a circuit pack. The ICN 486 provides the following features: address de-coupling; prioritized message delivery; dynamic destination transparency; active circuit pack selection; circuit pack level flow control; inter-network routing between separate media segments; and a broadcast capability for each media segment. The ICN supports the following physical media in the RDT: the common core shelf 26, the LAN, the access shelf ring gateway 58 and the NEP/SCP shared memory interface. The communication media in the Copper Access Shelf, the Auxiliary Shelf and the Fiber Access Shelf, as well as the communication media on the Fiber Access Passive Data Networks (PDNs), are not supported by the ICN. In the RDT, each circuit pack can act as a hub between each of the different media to which it is attached.

Detailed Description Paragraph Right (52):
In general, the circuit packs that exchange messages in the system are redundant and work in an active/standby scheme, in which the standby circuit pack is a warm standby. When a circuit pack in the transmission path of a message fails, protection switching from the active to the standby circuit pack is executed, along with such activities as fault location, reconfiguration, and fault recovery. The ICN handles transient intermittent communications faults that can be tolerated by retry and do not require protection switching. The ICN also provides notifications of permanent communications faults that demand protection switching in order to be processed by a circuit pack level or system level entity which can perform the required reconfiguration actions.

Detailed Description Paragraph Right (53):
The ICN de-couples the addresses used to deliver a message over the physical media by preferably providing a MID for every addressable node on the network. The ICN de-couples the queue identifier used to deliver a message to an application task on a circuit pack by providing ACIDs, which can be assigned to tasks at system startup.

Detailed Description Paragraph Right (54):
The ICN provides prioritized delivery of messages to an ACID. This provides the ICN with the ability to perform selective flow control in periods of high message traffic on a particular MID, as well as avoiding the priority inversion associated with the delivery of messages to a first-in-first-out (FIFO) message queue. The ICN provides dynamic destination transparency of the actual physical media being used to deliver a message. ICN provides a delivery option to allow an application to force the message to be delivered over a particular redundant media.

Detailed Description Paragraph Right (55):
When redundant circuit packs are available, the ICN provides transparency of the actual circuit pack providing service for a message. ICN provides a delivery option to allow an application to force a message to be delivered to a particular circuit pack. The ICN provides selective flow control based on priority for messages arriving to a circuit pack. The ICN broadcasts changes in flow control status to all nodes in the network. The ICN also provides inter-networking between the Common Core Shelf LAN, the Access Shelf Ring LAN and the NEP/SCP Shared Memory interface. A Broadcast Capability is provided on the Common Core Shelf LAN and the Access Shelf Ring LAN.

Detailed Description Paragraph Right (56):
The ICN message format that is delivered to the application consists of a ICN header 444 followed by a user data area indicated generally at 452 in FIG. 13. The ICN delivers a buffer to the user that starts at the user data area 452. The ICN header 444 includes source MID, source ACID, destination MID and destination ACID. The ICN functionality permits an application to post back a message without copying the message.

Detailed Description Paragraph Right (57):
The application software messages in the RDT preferably provide one of the following grades of service when delivering a message to another application: Non-guaranteed Message Delivery; Reliable Message Delivery; Reliable Message Delivery with Error Recovery; Guaranteed Message Delivery and Critical Guaranteed Message Delivery.

Detailed Description Paragraph Right (58):
Non-guaranteed Message Delivery service provides a single attempt to deliver a message using the non-guaranteed delivery option provided by the ICN. Use of this service is restricted. Examples of situations where this service can be used are messages that are internal to ICN or the common platform on a single circuit pack, messages that are generated and received by a circuit pack's bootstrap code and messages that are used by internal probes and monitors.

Detailed Description Paragraph Right (59):
Reliable Message Delivery with Error Recovery service provides a single attempt to deliver a message with no error recovery. The application sends the message once, and then the state data associated with the message is lost. Post backs are ignored. Application level responses and acknowledgements are also ignored. This service uses the guaranteed delivery option provided by the ICN. Since this service provides no error recovery, or retry timing, it is not used for operations which change the state, provisioned data or configuration of a circuit pack without some additional messages for verifying the state of the pack and/or rolling back inconsistent changes. This service is suitable for the delivery of protocol data units between application tasks which provide different layers of a protocol stack.

Detailed Description Paragraph Right (60):
Reliable Message Delivery service provides a single error recovery attempt to deliver a message with minimal error recovery. The application sends the message once and provides retries when a post back or recoverable send error is received.

Rèsponses and acknowledgements are optional. If a response of acknowledgement is expected, the application can provide retry timing and a finite number of retransmissions. The retry timeout and the number of retransmissions are fixed by system wide constants. This service is suitable for maintenance actions commands from an external managing system of craft user.

Detailed Description Paragraph Right (61):
Guaranteed Message Delivery service provides unlimited attempts to deliver a message. When using this protocol, all messages require a response message, e.g., either an acknowledgement or a response. The sending application must continually retry the message if no response is received within the application message delivery timeout. The application continues to retry the message until a response is received or there is an outage detected on the physical entity supporting the destination application. The retry timing is fixed by a system-wide constant. This protocol is preferably used for all provisioning messages between circuit packs and for all event messages announcing the failure of a facility or equipment, if that facility or equipment does not affect the system timing or internal network media.

Detailed Description Paragraph Right (62):
Critical Guaranteed Message Delivery service is a robust protocol that is used for critical messages which must be delivered. This protocol adds route diversity to the guaranteed message delivery protocol to ensure that a message is delivered. This service is used for all messages which must be delivered in a timely manner, even if the delivery media is in the midst of a reconfiguration or outage. This condition applies to events announcing a failure or recovery of a delivery media and commands to reconfigure the delivery media.

Detailed Description Paragraph Right (63):
With reference to FIG. 14, the user data area or application portion 452 of the message shown in FIG. 13 consists of three distinct parts. The first part is the inter-process message header (IMH) 446. The second part is the application header 448. The last part is the application payload 450. The application header and application payload are optional. An application header is generally required in order to provide some of the protocol services described above, such as guaranteed message delivery service. The inter-process message header (IMH) 446 is used by the application software to determine the format of the application header 448. This portion of the application message is generally fixed in size and format across the application software in the RDT. As described in connection with FIG. 12, the IMH 446 appears in the first bytes of the message after the ICN header 444. Some low-level messages can be defined to use just the IMH 446 and no application header 448, although applications using messages with only the IMH are restricted to the reliable message delivery protocol as the highest grade of service it can provide. Examples of messages using just the IMH include messages generated from interrupt handlers, messages that are internal to ICN or the common platform, messages that are generated and received by a circuit packs bootstrap code and messages that are used by internal probes and monitors.

Detailed Description Paragraph Right (64):
With reference to FIG. 15, the IMH 446 comprises the message class field 454, which is a one byte field that identifies the format of the application header. The following message classes are defined: (1) One class for each subsystem; and (2) cross subsystem messages which are not recognized as part of a subsystem. The message class is used to decouple the message formats of different subsystems from each other and from the common message format used across the subagent interface (SAI) 400, while allowing a single entity such as a fault tolerance task or a subsystem communication manager to perform a minimum amount of decode and processing on the message.

Detailed Description Paragraph Right (65):
The message identification field 456 in FIG. 15 is a one byte field that identifies the type of message being delivered. This field is present in the IMH to allow certain low level messages the flexibility of not defining an application header. It is recognized that in the interest of keeping the IMH to a reasonable size, this field may not provide enough address space to define all of the messages for a subsystem. A subsystem can use this field to define additional message categories

within a subsystem and reserve a field within the subsystem application header 448 for the actual message identifier. When not used, the reserved value for this field is "zero".

Detailed Description Paragraph Right (66):
The status code field 458 in FIG. 15 is the third field in the IMH 446. The status code is used in response messages to indicate the success or failure in the routing of the command. It does not reflect the status of the application command. Applications that do not use an application header can use this field for application oriented status. This field is divided into a user area and a system area. Status codes in the system area have the high order bit in the field reset. These status codes are reserved for system wide status codes. User status codes have the high order bit set and are available for applications to use. Possible status codes are as follows:

Detailed Description Paragraph Right (69):
3. Flow Control--The message could not be routed to the destination application due to flow control.

Detailed Description Paragraph Right (71):
The flags field 460 (FIG. 15) is the fourth field in the IMH and is one-byte long. The flags identify the type of service the message is to receive. They consist of the following: (1) Message Type (MT subfield)--this is a three bit subfield indicating the type of the message, which can be an Event (an unsolicited message), a Command (a solicitation for information or action), a Response (the result of a solicitation) or Unused (the message is not classified as to type); (2) Acknowledgment Requested (AR bit)--this informs the receiving application that this message must be acknowledged or responded to; (3) Acknowledgment (A bit)--This informs the originating application that this message is an acknowledgment to a previously received message; and (4) Retransmission (RTX bit)--this is a retransmission of a previous message. The table in FIG. 16 summarizes the allowed states of the message flags fields.

Detailed Description Paragraph Right (72):
With reference to FIG. 14, the content and format of the application header 448 is specific to the particular application message class identified in the IMH. All messages of a particular message class preferably use the same application header. The purpose of the application header is to support additional routing to objects or modules within the subsystem and to support the grade of service or services being employed by the subsystem. Examples of fields that can be present in the application header are object identifiers, transaction identifiers and sequence numbers. The content of the application payload is specific to the particular application message type.

Detailed Description Paragraph Right (73):
Protocol errors in the ICN will now be described. A recoverable send error is an error that is detected before the message is accepted at the entry to the ICN on the originating side. This type of error results in rejection of the message by the ICN. Examples of recoverable send errors are a circuit pack in flow control, a network outage or an attempt to send a message to a circuit pack that is not present in the network. An irrecoverable send error is a send error for which there is no recovery. Examples of these errors include formatting errors and resource exhaustion.

Detailed Description Paragraph Right (74):
An ICN post back occurs when a message is accepted at the delivery layer on the originating end and is subsequently undeliverable to the specified receiver. In this case, the message is posted back to the originator by an intermediary or ICN Fault Tolerance process which logs a message that cannot be routed by the ICN or sends a postback to the sender. The routing status code in the IMH indicates Invalid Route. Generally, only those messages with the AR bit set in the IMH are posted back.

Detailed Description Paragraph Right (75):
An application post back occurs when a message is delivered to the receiver and the receiving application subsequently determines that it cannot process the message. This occurs when an application declares a module, subcomponent, component or entire

subsystem in flow control or when an application determines that the entity being addressed is not present. Application flow control provides a subsystem with a higher level of flow control than the flow control offered by ICN, which is on a circuit pack basis only. For a subsystem, ICN flow control on a circuit pack (other than the NEP) is equivalent to flow control on a component. ICN flow control on the NEP is essentially equivalent to flow control being enacted simultaneously on at least one component, and at most the entire subsystem for each subsystem. If an application goes into application flow control, it stops accepting messages for the entity that is in flow control and posts all messages to that entity back to the originator. The routing status code 458 in the IMH indicates Application Flow Control. When an application is in flow control, it preferably must accept messages which release resources. The application typically accepts responses and acknowledgements while rejecting commands and events. When an addressed entity is not present, such as an object in an object base, the application posts the message back to the originator indicating Invalid Object ID in the routing status 458 code of the IMH.

Detailed Description Paragraph Right (77):
The Application I/O Timeout constant is used system-wide to time a response or acknowledgement to a message in all grades of service above the Non-guaranteed Message Delivery Protocol. The Application Error Recovery Timeout constant is used system-wide to time a delay for error recovery. This timeout is shorter than the Application I/O Timeout. The Max Application Retransmission constant is used to set the maximum number of times a message can be retransmitted after an application I/O timeout in the Reliable Message Delivery with Error Recovery protocol. The Maximum Network Delay constant is used to account for worst case network delay. It is used on the destination side of a transaction in the Guaranteed Message Delivery Protocols.

Detailed Description Paragraph Right (78):
With regard to the Non-guaranteed Message Delivery Protocol, this protocol can be used between a source application task and one or more destination application tasks. The protocol consists of a message sent from the source to the destination tasks. This protocol is generally used for event messages requiring an acknowledgement or commands requiring a response.

Detailed Description Paragraph Right (79):
The Reliable Message Delivery Protocol can be used between a source and destination application for message delivery with no error recovery above the delivery service. This protocol differs from the Non-guaranteed Message Delivery Protocol in that it uses the ICN guaranteed delivery option, and, as a result, generally cannot be used when sending a single message to multiple destinations. In this protocol, a message is sent and an application I/O timeout is initiated. Messages can have required responses or acknowledgements, but, if the response does not arrive within the application I/O timeout, the command is declared failed. All post backs and recoverable send errors cause the transaction to be declared failed immediately.

Detailed Description Paragraph Right (80):
The Reliable Message Delivery Protocol with Error Recovery provides error recovery to the reliable message delivery protocol. In this protocol, messages are sent and timed for at least one application I/O timeout. However, all recoverable send errors and post backs are retransmitted after a delay of one application error recovery timeout, as long as the time remaining the application I/O timeout is greater than the application error recovery timeout. As an option, messages for which a response or acknowledgement are expected can be retransmitted after each application I/O timeout, up to a maximum of max application retransmission times.

Detailed Description Paragraph Right (81):
The Guaranteed Message Delivery Protocol is essentially identical to the Reliable Message Delivery Protocol except that the number of retries for a message is infinite. With no post backs, send errors or response to a message, the original message is transmitted once every application I/O timeout. When a post back or send error is found, the message is transmitted once every application error recovery timeout, until the message is delivered successfully. This protocol continues to attempt delivery until either a response is received, the physical entity supporting

the entity being sent to experiences an outage (i.e. transitions to an OOS state), or the physical entity supporting the entity sending the message experiences an outage (i.e. there is no requirement to remember active transactions after an outage.) Every originating message in this protocol is preferably paired with an acknowledgement. Acknowledgements can be piggybacked onto a response, or they can be a separate message.

Detailed Description Paragraph Right (82):
The Critical Guaranteed Message Delivery Protocol adds route diversity to the service provided by the guaranteed delivery protocol. A message is sent on the primary media as usual.

Detailed Description Paragraph Right (87):
With regard to object organization, the RDT database 464 is organized as a collection of objects, and each subsystem maintains a subset of the entire collection. The entire database appears as one unified collection. Since there are no internal requirements on each subsystem as to how its data is organized or stored, the only visibility into this collection of objects is preferably through a common interface. This interface is the subagent interface 400. The subagent interface 400 allows an application to access the database by sending messages to an object instance. An application can do this without any regard for which subsystem contains the object instance.

Detailed Description Paragraph Right (116):
Once all contained objects have responded successfully to the deletion request, the targeted object responds to the originator of the message with a success indication. The originator of the message requests that the attempt be committed or de-committed. A de-committ can occur if the originator requires several deletes to be atomic, and one or more objects indicate they cannot perform the delete. Upon receipt of a committ or de-committ, the targeted object propagates the request to all contained objects. Once the targeted object has collected all successful responses to the request, it executes the request itself.

Detailed Description Paragraph Right (135):
Event announcement is the generation of messages from a circuit pack to the NEP 62 which announce the occurrence of a detected and verified error condition. Several announcements can be generated for a single event. Event announcement messages affect the automatic service state of a circuit pack or termination. The event announcement mechanism is via the guaranteed delivery protocol described above. Events must generally be verified before they are announced. Verification of an event can occur by a number of means. On a facility, the event is typically integrated for a period of time specified in the standard for that facility. Internally, an event can be verified by debouncing the input, retesting the input, or performing a non-service affecting local diagnostic.

Detailed Description Paragraph Right (136):
Event suppression is the filtering of duplicate announcements, and suppression of additional announcements which can be generated due to the failure of a supporting entity. The purpose of suppression is to avoid generating multiple event reports for the same overall condition. The event suppression mechanism is via propagation messages which update state information in the RDT database, and command messages which are delivered to circuit packs from an NEP subagent 414. Propagation messages occur between objects which are related using one of the relations described above. These messages occur according to object propagation policies.

Detailed Description Paragraph Right (137):
Event reporting is the process which can (1) update the standing conditions data store for the system; (2) update of the system alarm log; and (3) generate external alarm reports. The event reporting mechanism is via the subagent interface 400. All event report messages are generally sent to the Administrative Subagent object which represents the Administrative Subsystem 418. The Administrative Subsystem provides a repository for all event reports which are standing conditions, and manages the reporting of these events across a variety of interfaces. These interfaces include the following: (1) active sessions on the local asynchronous interfaces (sessions can be active for the following types of users: Craft; NMA; and RIDES); (2) system

front panel; (3) E2A SAC; (4) E2A APR; (5) CO Audibles and Visuals; and (6) TR-TSY-000008 Derived Data Link. Additionally, the Administrative Subsystem is responsible for maintaining and determining the system alarm state. To support alarm retrieval requirements, the Administrative Subsystem provides an interface to retrieve all standing conditions for a particular object in the database.

Detailed Description Paragraph Right (146):
The Application Identifier (AID) data type identifies which software made the logging entry. AID is defined below in section 5. As shown in FIG. 20, the Application Identifier identifies a software element according to its position in the RDT software architecture hierarchy, as opposed to its runtime parameters such as the program counter and process identifier. The requirements for logging a process identifier and program counter were removed from the RDT. A process identifier will not indicate much on subsystems which define tasks to be nothing more than an execution shroud. Subsystems which provide functionality that is always associated with one task designate these tasks as components 470, subcomponents 472 or modules 473. This indicator can then be permanently associated with a particular task. The Subsystem portion 468 of the Application Identifier is defined as a system-wide constant. This is the same set of constants that are used in the IMH message class field. The remaining portions are defined in the subsystem architectures.

Detailed Description Paragraph Right (161):
Level 1--Level 1 trace statements indicate the begin and end points of normal transaction processing. A level 1 trace log is generally an indication of all inter-circuit pack messages. The collection of level 1 trace logs across the system is typically a reliable test of conformance to behavior specifications.

Detailed Description Paragraph Right (162):
Level 2--Level 2 trace statements capture all inter-subsystem messages and transactions within a circuit pack. On the NEP, a trace log at level 2 generally indicates all SAI transactions and their returns, as well as all level 1 log entries.

Detailed Description Paragraph Right (163):
Level 3--Level 3 trace statements capture intra-subsystem messages. Care is taken to restrict tracing to a small set of components at trace level 3 and above. This trace level captures all send and receive calls to ICN and the SAI whether they are used between subsystems or internal to a subsystem. Level 3 trace statements generally also indicate high level logic flow in a subsystem. Entries and exits at the component or sub-component level are traced.

Detailed Description Paragraph Right (164):
Level 4--Level 4 trace statements capture resource usage in a software component. Trace statements at this level indicate memory resource allocation and de-allocation, semaphore usage, task allocation and timer usage. This include buffers used to send and receive messages and data buffers used for the long term storage of data records or objects within the subsystem. Level trace statements generally indicate mid level logic flow in a subsystem. Entries and exits at the sub-component or module level are traced.

Detailed Description Paragraph Right (173):
As previously discussed, the RDT database 464 is organized as a collection of objects that are distributed across subsystems. The subagent interface (SAI) 400 is the interface that is used to tie this distributed database into a virtual machine interface. This interface is preferably used for all communication between agents and subagents on the NEP. In addition, subsystems can use the SAI for intra-component communication within an agent or subagent on the NEP. The SAI provides the following functions: (1) construction and manipulation of messages; and (2) application level delivery of messages.

Detailed Description Paragraph Right (174):
The interface 400 is organized as an message-based interface. A subsystem component on the NEP communicates with another subsystem component by sending a message to one of the objects contained by that subsystem. Subsystems perform operations on these

objects by sending them messages. The SAI directs messages targeted to objects to the appropriate subsystem for processing. This is accomplished by requiring that each subsystem provide a name resolution function that the interface can call. This function is essentially a look-up table for the objects in a subagent. A subagent can have many more object instances than the ones that are visible across the SAI 400. Only the objects that are visible across the SAI, however, need be represented in the name resolution function. As shown in FIG. 22, the SAI is a value-added interface which resides in the NEP only and performs higher level messaging between subsystems than the messaging provided by the ICN of the System Services Subsystem 416, which provides messaging, for example, between microprocessors or between processes associated with two different circuit packs.

Detailed Description Paragraph Right (175):
This abstraction is convenient to the subsystem that is performing an operation because it hides implementation details and differences between subsystems that provide similar services. For instance, the RDT can provide several types of access shelves, each with objects representing POTS subscriber terminations. A subsystem requiring access to these subscriber terminations does not need to know how each access shelf subsystem implements these objects or, in other words, without knowing in what subagent a particular object resides. A subsystem operates on a particular object instance by knowing (1) the identifier for the object, and (2) the messages available for that object class. The identifier for the object is called the Subagent Interface Object Identifier (SAOID). This is a unique name for the object. The naming scheme is similar to TL1 AIDs in that it is based on the physical architecture of the RDT. However, the SAOID also contains an identifier of the subsystem which contains the object instance. This is how a request is routed to the correct subsystem.

Detailed Description Paragraph Right (176):
The message identifies a service or event and its associated parameters. The underlying organization of the objects in the database (for example, which subsystem contains a particular object instance, and how the operation is carried out) is hidden to the subsystem sending the message. By describing the set of messages associated with each object class and the mechanism for delivering messages, the interface 400 is defined.

Detailed Description Paragraph Right (177):
The SAI 400 interfaces to each of the subsystems through a registered name resolution function. The purpose of the name resolution function is to provide subsystems the flexibility of having messages routed to a particular task based on the SAOID for which the message is targeted. The name resolution function registration is accomplished statically before system build time. As stated previously, the SAI uses the ICN component of the System Services Subsystem to deliver messages.

Detailed Description Paragraph Right (179):
There are a number of requirements for subagents interfacing with the Subagent Interface 400 and subagents providing services in support of the Subagent Interface. Each subagent must export a name resolution function. Each subagent must process all messages defined for an object class if the subsystem supports that object class. A list of the object classes that must be supported for each subsystem is provided in FIGS. 23 through 31. All messages across the subagent interface are preferably delivered with the above-described Reliable Message Delivery Protocol with Error Recovery.

Detailed Description Paragraph Right (181):
The subagent interface object classes define the set of services that are available in the database. These services are grouped according to the entity they are performed on and, therefore are described as objects. For example, for each circuit pack in the system, the network management agents are able to remove the pack from service, restore the pack to service, reboot the pack and retrieve its CLEI code. The interface object class SAICircuitPack defines these services: (1) remove--remove the pack from service; (2) restore--restore the pack to service; (3) restart--reset the pack; and (4) retrieve--retrieve information about the pack. Each service maps onto one message in the implementation. This mapping is at least 1:1 but it can be

1:n because optimizations are available (i.e., it is possible to accomplish several services with one message definition.) These messages define the interface through which the network management agents perform these operations on any circuit pack in the system, despite the fact that some circuit packs are managed by the FAS subagent, some are managed by the CCS subagent and some are managed by the TAS subagent. Further, each of these subagents can have slightly different representations of the data and different implementations of the services. The SAI 400 provides the following services:

Detailed Description Paragraph Right (188):
The SAI Equipment defines messages to access and operate on physical equipment object in the RDT database. The definition includes messages to manipulate the service state of an equipment and access its contact points.

Detailed Description Paragraph Right (189):
The SAI Equipment Molder defines messages to access and operate on physical equipment slot objects in the RDT database. The definition includes messages to provision an equipment slot for a particular circuit pack and retrieve the equipment status from the slot.

Detailed Description Paragraph Right (190):
The SAI Circuit Pack defines messages to access and operate on circuit pack objects in the RDT database. The definition includes messages to boot the pack and restart the pack.

Detailed Description Paragraph Right (191):
The SAI Memory defines messages to access and operate on different types of memories in the RDT. The definition includes messages to backup and restore a memory. All instances of memory objects are contained in the Administrative Subsystem 418.

Detailed Description Paragraph Right (192):
The SAI Network Element defines messages to access and operate on the network element object in the RDT database. This object is contained in the Administrative Subsystem 418, and there is only one instance in the RDT. The definition includes messages to set and retrieve system wide attributes such as the system timing sources, system name and system time and date.

Detailed Description Paragraph Right (193):
The SAI Virtual Network Element defines messages to access and operate on the virtual network element objects in the RDT database. These objects are contained in the Administrative Subsystem 418 and there are up to 24 instances in the RDT. The definition includes messages to create and delete terminals and add and delete feeders and binding posts.

Detailed Description Paragraph Right (197):
The SAI Cross Connection object class defines messages to create, delete, edit and retrieve cross connects. All cross-connects are contained in the Call Processing Subsystem.

Detailed Description Paragraph Right (199):
This object class represents the Network Managing Subsystem 420. It is contained by the Network Managing Subsystem, and there is only one instance. The definition of this object defines messages to advance the subagent through the subagent state model.

Detailed Description Paragraph Right (200):
This object class represents the AUX Shelf Subsystem. It is contained by the AUX Shelf Subsystem, and there is only one instance. The definition of this object defines messages to advance the subagent through the subagent state model.

Detailed Description Paragraph Right (201):
This object class represents the Call Processing Subsystem 422. It is contained by the CP subsystem 422, and there is only one instance. The definition of this object defines messages to advance the subagent through the subagent state model.

Detailed Description Paragraph Right (202):
This object class represents the Transport Access Shelf (TAS) Subsystem 404. It is contained by the TAS Subsystem 404, and there is only one instance. The definition of this object defines messages to advance the subagent through the subagent state model.

Detailed Description Paragraph Right (203):
This object class represents the Common Core Shelf Subsystem 406. It is contained by the CCS subsystem and there is only one instance. The definition of this object defines messages to advance the subagent through the subagent state model.

Detailed Description Paragraph Right (204):
This object class represents the Fiber Access Shelf (FAS) Subsystem 408. It is contained by the FAS Subsystem 408, and there is only one instance. The definition of this object defines messages to advance the subagent through the subagent state model.

Detailed Description Paragraph Right (205):
This object class represents the Turnup Subsystem 424. It is contained by the Turnup Subsystem 424, and there is only one instance. The definition of this object defines messages to advance the subagent through the subagent state model.

Detailed Description Paragraph Right (206):
This object class represents the system Alarm Log. It is contained by the Administrative Subsystem 418, and there is only one instance. This object class provides messages to retrieve the log.

Detailed Description Paragraph Right (207):
This object class represents the Error Logs on each circuit pack performing logging. There is one instance per error log in the system and the instance is contained by the subsystem managing the circuit pack whose error log it represents. This object class provides messages to retrieve the contents of the log and initialize the log.

Detailed Description Paragraph Right (208):
This object class represents the Trace Logs on each circuit pack performing logging. There is one instance per trace log in the system and the instance is contained by the subsystem managing the circuit pack whose trace log it represents. This object class provides messages to retrieve the contents of the log, initialize the log and set the trace level.

Detailed Description Paragraph Right (209):
This object class represents DS1 lines that terminate on the feeder side of the system. This object class defines messages which are used to configure the DS1 line.

Detailed Description Paragraph Right (210):
This object class represents DS1 lines that terminate on the feeder side of the system. This object class defines messages which are used to configure the DS1 line and also perform binding post assignments.

Detailed Description Paragraph Right (211):
This object class represents the local asynchronous ports on the MAU. There are four instances of this object class contained by the Network Management Interface Subsystem 420. This objects class defines messages to send to and receive from a local asynchronous port. The use of these messages are restricted to the Administrative Subsystem 418 and the Network Management Subsystem 420.

Detailed Description Paragraph Right (212):
This object class represents a system wide schedule for backing working memory up to non-volatile memory. There is one instance of this object, and it is contained in the Administrative Subsystem 418. This object defines messages to create, delete, edit and retrieve the schedule.

Detailed Description Paragraph Right (213):
This object class represents a system wide schedule for performing diagnostic tests.

There is one instance of this object, and it is contained in the Administrative Subsystem 418. This object defines messages to create, delete, edit and retrieve the schedule.

Detailed Description Paragraph Right (215):
This object class represents a TL1 user session. This object class is created when the user activates a session and deleted when the user cancels the session. This object defines messages to validate TL1 commands, inhibit and allow messages on the session.

Detailed Description Paragraph Right (225):
Regarding the NEP Agent/Subagent, each subsystem, with the exception of System Services Subsystem 416, generally contains an application component targeted for the NEP. These components, which are indicated generally at 416 and 424 in FIG. 5, comprise a set of application components on the NEP. These components are agents or subagents. These components have the following characteristics: (1) provide support for Subagent Interface 4000--the agent or subagent component provides support for the SAI to the subsystem, including receiving and processing all SAI messages directed to the subsystem and providing routing support for SAI messages; and (2) provide RDT database domain--the agent or subagent component allocates and manages the working memory for the database objects in the subagents domain.

Detailed Description Paragraph Right (228):
The Quiet State 496 is the initial state for an agent or subagent. In this state, the subagent exists with a minimum of resources and it does not generate any external ICN message traffic or subagent interface transactions.

Detailed Description Paragraph Right (235):
The operational active state 510 is a fully initialized state in which the subagent is providing its full functionality. In the RDT, all agents and subagents on the in-service, active NEP are in the operational active state. The current state of the subagent is the system state and can be imposed on any supported entities in the Operational Active state 500. Transitions can occur from either the Operational Cold or Operational Standby states. In this state, the subagent receives all of its ICN and external message traffic and generates all of its ICN and external message traffic. The behavior exhibited by the agent or subagent is to receive a request or message and process it. Subagents in this state are generally required to provide synchronization messages to the mate NEP as internal subagent attributes are modified or at predefined check points.

Detailed Description Paragraph Right (244):
The primary purpose of the Network Management (NM) Subsystem 420 (FIG. 4) in the RDT is to support communication with external managers for the purpose of controlling and monitoring the RDT. With reference to FIG. 36, the architecture of the subsystem comprises three main modules, that is, a CMISE-based network management agent 524, a TL1-based network management agent 526, and the Management Information Base (MIB) 528, and a set of communications interfaces. The MIB 528 is logically located outside of the NM Subsystem 420 because it must be accessed by other subsystems in the RDT, but is nevertheless an important part of the NM Subsystem 420. FIG. 37 depicts the software hierarchy of the NM Subsystem. The subsystem can be split into two pSOS tasks. pSOS refers to an operating system which is commercially available from Software Components Group, San Jose, Calif. The Upper Layer Task denoted generally at 530 comprises the CMISE Agent, the TL1 Agent, and the upper layers of the communication interfaces. The Lower Layer Task denoted generally at 532 comprises the necessary protocol layers for reliably transporting messages from a remote manager to the RDT.

Detailed Description Paragraph Right (246):
Threading relates to supporting multiple execution flows without requiring separate tasks. There are two basic types of threads in the NM tasks: service execution and command execution. Multiple service threads can be carried out simultaneously by building state machines for each control path. The state machine contains the current state information of the thread and an automata to determine the valid next states, based on events, that the thread can transition to. As the thread executes, its state machine is updated to reflect its current state. A thread can suspend

itself by sending out a message and transition to a "wait" state for the reception of a response. If the thread suspends, all information about its state before suspension can be retrieved from the state machine and execution can be continued upon resumption. Service threads are used in the Lower Layer Task 532 for supporting multiple protocol stacks and multiple associations per stack. When a message is received from a manager containing an association, it is routed to the correct stack based on a destination identifier in the message. The stack machine, in turn, calls the protocol state machine to determine how to react to the message for the particular association it has appeared on. If the current state of the association indicates that the message is valid, the machine transitions to a new state and executes operations based on the message. If the message is invalid (e.g. a data request on a terminated association), then the stack generates an error. The other type of thread is for command execution, as described in further detail below. Since the RDT supports multiple simultaneous commands from multiple external managers, it creates a command thread to track the execution of the command within the RDT. The command thread sets up a state machine with the current state of the command (e.g. parsing, send to Object Request Broker (ORB), waiting for completion status, building response) and commences processing. Command threads are essentially contained completely within the two agents 524 and 526.

Detailed Description Paragraph Right (247):
The basic functionality of both agents 524 and 526 is to coordinate command execution. Each agent receives a command from an external manager and creates a thread to track the command processing. Next, the agent simplifies the command into a sequence of discrete operations and sends each discrete operation to the Object Request Broker for execution. Since the agents and the ORB are in separate processes, a pSOS message needs to be sent to the ORB for the operation request. While the MIB is a passive element for storing information, an ORB is an active software element that is partitioned in relation to the subagents. Each ORB partition preferably accesses a particular part of the MIB. The ORB and the MIB, however, can be combined or implemented separately. After the message is sent to the ORB, the agent suspends the command thread and executes others, until the ORB sends a callback message detailing the execution status and supplemental information, as indicated by the arrow labelled Internal Event Notification and Remote Operations Callback in FIG. 36. The agent generally does not suspend completely after the request because it can be a relatively long time before the ORB sends a response. In the meantime, the agent continues processing other threads. For example, the ORB may need to set data on a peripheral card on a channel shelf 28 before completing the operation. This mode of operation is termed "Asynchronous Request and Callback". Once the execution thread has completed and built the appropriate response messages to the external manager, it terminates itself, and the corresponding resources are returned to the pool of resources for the system. In the diagram, the arrow labelled Remote Operations Request to the ORB can be seen extending from each agent. The callback comes into a message queue and is routed to the correct agent via the Message Router 534, as shown in FIGS. 38 and 39.

Detailed Description Paragraph Right (249):
Along with command execution, the NM Subsystem 420 supports the capture and transformation of internal event notification messages into CMISE M-EVENT-REPORT and TL1 Automatic messages for transmission to external managers. The internal event notifications are placed in an Upper Layer Task input queue and routed directly to both agents 524 and 526 by the Message Router 534. Each agent has the knowledge of how to format the notification into the appropriate standard (CMISE or TL1) event message and upon what channels to place the formatted event message. Once an agent formats an event message and determines the communication path to send it over, it calls the correct stack implementation to carry out the transmission.

Detailed Description Paragraph Right (251):
The communications architecture is broken into two pSOS tasks: an Upper Layer Task 530 containing the agents and upper stack layers, and the Lower Task 530 containing the lower stack layers. The two tasks communicate using queue mechanisms supported by the ICN 486. Individual modules in each task package an outgoing message into a message, and call the ICN to send it to the correct destination. A message router 534 and 536 in each task reads a message from the incoming queue, strip off the ICN header 444, and calls the correct service thread to process the message.

Detailed Description Paragraph Right (253):
The Lower Layer Task 532 is primarily responsible for the reliable transportation of messages between the RDT and a set of external peer entities. It contains the Transport, Network, and Datalink modules that receive, reassemble and error-check messages for delivery to the NM Upper Layer Task 530 or the Call Processing Subsystem 422. It communicates with the NM Upper Layer Task above and with the Spyder-T interrupt handlers 539 below.

Detailed Description Paragraph Right (254):
The four remote protocol stacks and the local craft interface will now be described. A TL1/X.25 stack from a Remote Craft Device or an OS is provided, as indicated in FIG. 36 by an arrow labelled TL1Message-SS. This stack uses the X.25 module 538 in the Lower Layer Task to reliably receive TL1 commands and pass them, via the Message Routers 536 and 534, to the TL1 Convergence Function 540. The TL1 Convergence Function handles connection establishment, release, and aborts; otherwise, it sends a complete TL1 command message to the TL1 Agent 526 for parsing. As shown in FIG. 38, the TL1 Parser module 542 analyzes the input message for syntax and minor semantic errors, while building an agent request bound for the ORB 529. If parsing is successful, the resulting agent is sent. If not, the Parser calls the TL1 Formatter module 544 to build a error response in TL1, and sends it back down the TL1/X.25 stack, that is, TL1 Convergence Function 540, Message Routers 534 and 536, X.25 module 538 and then to the remote manager.

Detailed Description Paragraph Right (255):
An OSI 7-layer stack for the Supervisory System and testing purposes is provided, as indicated in FIG. 36 by an arrow labelled CMISE Message-SS. This stack uses the X.25/LAPB and Transport modules 538 and 548 in the Lower Layer Task 532 to reliably receive CMISE commands and pass them to the Session module 550 in the Upper Layer Task 530. From there, the message is transmitted to the CMISE module 552 and converted to a CMIP Interface Data Unit structure accepted by the CMISE Agent. A CMIP Interface Data Unit, referred to hereinafter as a CMIDU, comprises all of the information in the CMISE request, but formatted to a form more readily processed and understood by the CMISE Agent. While the CMISE message is converted to a CMIDU in the protocol stacks, the Presentation, ROSE, and CMISE modules 554, 556 and 552, respectively, perform syntax checking on the CMISE message. If the message is found to be valid and correct, the CMIDU is sent to the Agent 524 for processing. If not, the module that detects the error builds an error response, and sends it back down the stack without the CMISE Agent being aware that a request was received. It does not include security or semantic validation of the message, which is done by the Agent.

Detailed Description Paragraph Right (256):
An OSI short stack based upon Section 126.1 of TR-TSY-000303 is provided and is illustrated in FIG. 36 by an arrow labelled CMISE Message-Switch. This stack uses the LAPD module 546 and the TR-303 Convergence Function module 554 in the Lower Layer Task to receive the CMISE command. While this implementation of the TR-303 Convergence Function performs the same functionality as that defined in TR-303, it is constructed differently. The TR-303 Convergence Function then passes the CMISE message to the Session module 550 in the Upper Layer Task 530. The rest of the flow is the same as the 7-layer stack implementation.

Detailed Description Paragraph Right (257):
A SONET 7-layer OSI stack is provided which is defined in TR-253. The lower three layers, Physical, LAPD, and the Connectionless Layer Network Protocol (CLNP) are supported on the SOP circuit pack 63. Network layer routing software is based on the ISO ES-IS standard. LAPD is connected to CLNP using the Subnetwork-dependent Convergence Function (SNDCF) software developed by Retix Incorporated, Santa Monica, Calif. The SOP circuit pack sends a Transport Layer message to the Transport module 548 of the Lower Layer Task, as indicated by an arrow labelled CMISE Message-SOP. The remaining paths up the CMISE stack are the same as the CONS-type OSI 7-layer stack.

Detailed Description Paragraph Right (258):
The Local Craft Interface is provided via a UART on the Maintenance cirucit pack, as

indicated by an arrow labelled TL1 Message-LCI. The UART is controlled by a handler task on the Maintenance circuit pack which provides enough dialogue to receive a complete TL1 command and package it in an ICN message. The ICN transmits the message to the Upper Layer Task 530 where it is routed directly to the TL1 Parser module 542. As with the TL1/X.25 stack, if the Parser is successful it sends an agent request to the ORB; if not, it calls the TL1 Formatter 544 to send a TL1 error response back to the local craft interface. This communication channel does not use the Lower Layer Task 532 at all.

Detailed Description Paragraph Right (260):
The CMISE external message, encoded in ASN.1 and adhering to an international standard, is primarily translated in the CMISE module 552 (FIG. 36) of the Application Layer to a CMIDU. All information in a CMISE message is fully parsed and validated, except for managed object attribute data values. Naming, filtering, and attribute list parameters are examples of fields that contain attribute value data. CMISE passes attributes values to the CMISE Agent in their encoded form. All other information is parsed. If CMISE discovers an error in parsing the received message it generate an error reply and return it to the manager without alerting the agent that a CMISE message was received. For output from the CMISE Agent, complimentary steps are taken. The Agent builds a CMIDU containing internal data except for attribute values to ASN. 1 which it will have to encode explicitly and place in the CMIDU.

Detailed Description Paragraph Right (261):
The reason the CMISE Agent is responsible for encoding/decoding individual attributes is it eliminates the need for bulk encodes and decodes on areas of the original CMISE message that may never be used. The Agent decodes an attribute value generally only once during the processing of a CMISE operation, no matter how many times the attribute is accessed. The Agent encodes an attribute generally only once, no matter how many times it is duplicated in the response or event.

Detailed Description Paragraph Right (264):
2. Name Resolution (synchronous)--Using the Name Resolution Interface 560, the Core converts the Distinguished Name of the base managed object defined in the CMISE message to an internal reference called an Instance.sub.-- Ref.

Detailed Description Paragraph Right (269):
8. Suspended thread (asynchronous)--The Agent Core 558 suspends this execution thread until the ORB 529 sends a callback message containing the completion status of the current operation.

Detailed Description Paragraph Right (270):
9. Receive Callback (asynchronous)--The Core 558 receives the execution callback containing the completion status and builds a CMISE response structure. There is one response structure per instance. If the CMISE request works on more than one managed object instance, the Agent generates individual CMIDUs containing a response for each instance, and correlates them using the link-reply parameter in the CMIDU. The CMISE Agent converts each CMIDU into a CMISE message and sends it to the correct manager.

Detailed Description Paragraph Right (274):
Notifications of internal events are received in an asynchronous manner by the CMISE Agent 524. Generally, the only responsibility of the agent is to report the event using a standard M-EVENT-REPORT message to necessary external CMISE-based managers. All internal behavior initiated in response to the event (e.g. facility protection switch or equipment switch) is typically carried out by other subsystems. More than one manager may want to know about the event. The agent determines which of the managers to send the event to by checking the state and authorization status of the manager held in the MIB 528.

Detailed Description Paragraph Right (279):
The goal is for the CMISE Agent to be a structured automata feeding off the CMISE and AO areas of the Schema in developing operations messages to the ORB. In this way, the CMISE Agent code is compact and generic. Extensions to the external interface can be provided via table updates to the Schema.

Detailed Description Paragraph Right (283):
The TL1 Agent performs the same basic function as the CMISE Agent but for the TL1 commands. There are two major functional differences between the TL1 Agent and the CMISE Agent: (1) where the external message is translated; and (2) how the MIB Containment Tree is referenced.

Detailed Description Paragraph Right (284):
A CMISE external message, encoded in ASN. 1 and adhering to an international standard, is primarily translated in the CMISE module 552 of the Application Layer to a CMIDU. A TL1 message is ASCII-based and very simple. Thus, it does not need complex translation procedures. The TL1 Agent 526 performs all of the parsing of TL1 commands, and the formatting of TL1 responses.

Detailed Description Paragraph Right (293):
Notifications of internal events are received in an synchronous manner by the TL1 Agent. As with the CMISE Agent, the only responsibility is typically to format a TL1 automatic message, and send it to the proper manager and local craft device. The TL1 agent does this by applying a policy defined in the MIB for the routing of TL1 events.

Detailed Description Paragraph Right (307):
In this example, the following parsed TL1 command is mapped to a TL1 proxy object and ultimately to an SAI message: ENT-TO::::TypE=:gsfN=4DU, RATE=24, SC=N. The Schema tables associated with the TL1 Agent 526 provide the information listed in the tables depicted in FIGS. 40, 41, 42 and 43. As shown in FIG. 40, the command (ENT-TO) and the parameter (GSPN=4DU) correspond to a message ID XI.sub.-- DEF.sub.-- MSG.sub.-- ID.sub.-- DLT.sub.-- SET.sub.-- C). With reference to FIG. 41, the message ID is preferably converted to the one byte field 456 (FIG. 15) by using a look-up table (not shown) to match the message ID to a binary number between 0 and 255. Further, the SAI service and the object class are also derived from the table, depicted partially in FIG. 40, based on the command, and corresponding modifiers, and, in some cases, parameters.

Detailed Description Paragraph Right (308):
The TL1 AID is combined with the object class to obtain an SAOID (FIG. 42). Further, the parameters in the TL1 command are converted to SAI attributes using a table that is partially depicted in FIG. 33. The combination of the SAOID, the message ID, and the SAI attributes is sufficient to generate an SAI message, in response to the TL1 command in accordance with the present invention, and to therefore invoke service to operate on objects in the MIB. This is because objects in the MIB have an address (SAOID), a name (object class) and a list of attributes such as parameters that can be modified in the object.

Detailed Description Paragraph Right (318):
The Auxiliary Shelf or Test Subsystem 410 comprises the Auxiliary Shelf 608 Subagent and an interface to the CCS subsystem which allows the test subagent to terminate the TR-008 DDL test messages. The Auxiliary Shelf Subsystem performs initiation and coordination of TR-TSY-000008 channel and customer loop testing, and craft initiated channel and customer loop testing for the RDT. In addition, the Auxiliary Shelf Subsystem manages the operation of the circuit packs in the auxiliary shelf.

Detailed Description Paragraph Right (323):
The System Services Subsystem 416 provides the operating system functions, the internal message delivery services and the interfaces to standard components, such as non-volatile memory for the RDT. In addition, this subsystem provides standard functions for software error detection. This subsystem is structured as a collection of cohesive services with low level coupling between them. This subsystem, or a portion of it, is resident on the circuit packs. Each service provided by the subsystem therefore must generally be configured for each environment. The following services are provided by this subsystem are (1) Real Time Operating System Extension (RTOSE); (2) Internal Communication Network (ICN); (3) error envelope; (4) watchdog timer service; and (5) Timed Services Manager.

Detailed Description Paragraph Right (326):

The RTOSE Signal management services 616 manage operating kernel signal objects. These objects include events, semaphores, and asynchronous signals. The RTOSE Queue Management services manage operating kernel message queues. The RTOSE Memory Management services manage operating kernel memory resources. The RTOSE Timer Management services allow a user to start and/or stop a timer block object. The timer resolution is at least 100 ticks/second.

Detailed Description Paragraph Right (327):
The Internal Communication Network (ICN) 486 interconnects the processors of the common shelf, the auxiliary shelf, and the access shelves. ICN connections are provided as a message delivery network. The sender need only know the receiver's node name and queue name for a message to be delivered correctly.

Detailed Description Paragraph Right (339):
The Subagent Interface (SAI) is used for all subsystem communication on the NEP. It provides an abstraction that permits information about where and how operations are performed to be hidden from the applications. In addition, the interface provides a uniform view of system entities which is independent of the internal organization of each subsystem. The SAI provides the following functions: (1) constructing and manipulating of SAI messages; and (2) object-based communication.

Detailed Description Paragraph Right (340):
The SAI interfaces to each of the subsystems through a registered name resolution function. The purpose of the name resolution function is to provide subsystems the flexibility of having messages routed to a particular task based on the SAOID the message is targeted for. The SAI interfaces to ICN through a public API, and comprises two major components (1) SAI Message Component 632, and (2) SAI Object Communication Component 624, as shown in FIG. 55, which depicts the hierarchy of SAI components.

Detailed Description Paragraph Right (341):
The SAI message components 632 is an object class library that facilitates the construction and manipulation of SAI messages. SAI messages are messages between subsystems that represent a request, response, acknowledgment, or event. SAI messages are sent to objects defined by the interface object classes associated with the software architecture via the SAI object communication component 634.

Detailed Description Paragraph Right (342):
The object communication component (OCC) 634 is a library which encapsulates ICN and delivers messages created by the message building component 632 to the correct subsystem task queue based on the object instance identifier (SAOID) to which the message is targeted. The object communication component also receives SAI messages arriving at a subsystem filtering out non-SAI transactions. By allowing applications to direct their messages toward objects rather than specific tasks, the details about which subsystem is responsible for handling a particular message and the internal organization of subsystem internals is hidden.

Detailed Description Paragraph Right (343):
FIG. 56 depicts the SAI components 632 and 634 within the context of the system. The sending task 636 uses the message class library to construct and address the message and initialize the message payload. The message class library invokes the ICN 486 when the message is created to allocate a buffer for the SAI message. The sender then uses the object communication component 632 to send the message to one of the interface objects, for example, a DS0 channel termination. The object communication component determines which subsystem and ACID within the subsystem to route the message to using the SAOID and a registered name resolution function. The message is then delivered to the ACID via ICN. The receiving task calls the object communication component 632 to receive SAI messages. The receiving task can alternatively call ICN directly to receive the SAI message. This requires the receiving task to perform a decode of the message class in the IMH header before using the SAI message component to access the message payload. When the message is received, the receiving task uses the message class library to determine the message type and access the message payload. The method of sending the response back to the sending task is essentially the same. The receiving task becomes the sending task and the sending task becomes the receiving task.

Detailed Description Paragraph Right (345):
The SAI uses the memory management scheme imposed by ICN with added protocols that make the originator of the message responsible for allocation and de-allocation of a message object. One exception to this rule is unacknowledged event messages. If an event message is sent across the SAI and an acknowledgment is not requested, the receiver is responsible for deleting the event message. Allocation of the ICN message buffer is encapsulated by the message component of the SAI. When a message is sent, the sender has no rights to the message and cannot delete it.

Detailed Description Paragraph Right (346):
Applications generally must specify a target object address (SAOID) to initiate communication with entities outside of their subsystems. The object address comprises components shown in FIG. 57 which are preferably organized as a 32 bit entity. A class is provided to construct and manipulate SAOIDS and to therefore allow future implementation to alter the structure and format of this address. The SAI object communication component 632 determines the subsystem to route the message to based on the subsystem type component 638 of the SAOID. On determining the correct subsystem, the SAI calls a statically registered name resolution function supplied by each subsystem to resolve the destination ACID of the message. This function takes an SAOID as an argument, and returns an ACID. The ACID is used to identify a task queue which the ICN uses to deliver the message. Subsystems wanting all SAI transaction to arrive at one queue register a name resolution function that returns the same ACID regardless of the SAOID passed to it.

Detailed Description Paragraph Right (347):
SAI messages comprise the following return address information: (1) from ACID; (2) from SAOID; and (3) transaction ID. This return address information is used by the SAI to return responses and acknowledgements back to the original requestor 636. Applications responding to or acknowledging SAI transactions do not need to specify address information.

Detailed Description Paragraph Right (348):
Each SAI command message has a response. SAI command and response message payload formats are identical. The flags field in the IMH header 448 indicates whether the message is a command or a response. The receiver of an SAI command fills out the response fields contained in the command message and sends the message back to the original sender. Likewise, acknowledgments to SAI event messages are identical to the original event message with the flags field set in the IMH header indicating an acknowledgment. Since SAI messages are generally always routed on the NEP, there is no additional overhead incurred using this scheme. In addition, the receiver does not need to de-allocate each SAI message and re-allocate a response or acknowledgement message.

Detailed Description Paragraph Right (349):
Because some object attributes can only be set once during creation and defaulting may occur, object creation messages are distinguished from object modification or "set" messages. Object creation messages are addressed to the object identifier (SAOID) to create. These messages contain values for all semantically related object attributes. Defaults for attributes not supplied by the application are provided where appropriate. In addition, each attribute has an associated "status" attribute which is used to identify detail status information when create operations fail. Object deletion messages are addressed to the object identifier (SAOID) to delete.

Detailed Description Paragraph Right (350):
Modification of object attributes is accomplished by sending one or more "set" messages to an object. As a rule, all semantically related attributes are grouped together in one set message. Unique set messages can be developed to modify other logical groups of attributes, such as performance thresholds and alarm severity assignments. In addition, attribute modification operations which have special behavior or imply a higher level operation are accomplished with "action" messages. For example, "remove" or "restore" operations are used to bring entities in and out of service rather than "set" operation on the primary service state. Each attribute in a set message has an associated "status" attribute to identify detail status information when set operations fail, and to indicate to the receiver of the message

the attributes being modified.

Detailed Description Paragraph Right (351):
Retrieval of object attributes is accomplished by sending one or more "get" messages to an object. As a rule, "get" messages are similar to the "set" messages but include additional attributes which can not be modified. As with "set" messages, unique messages can be developed to retrieve other logical groups of attributes such as performance monitoring information and current problem lists. Operations which perform actions on objects comprise all of the attributes necessary to perform the operation.

Detailed Description Paragraph Right (352):
The SAI object communication component filters messages for synchronous send and receive operations. On synchronous blocking send operations, the object communication component waits for the response to the originating transaction and filters out all other responses, placing them on an error queue specified by the application. On receive operations, the object communication component waits for SAI transactions to arrive. All other arriving transactions generate an exception.

Detailed Description Paragraph Right (353):
As discussed previously in connection with FIG. 12, an SAI message preferably comprises four distinct parts. The first part is the ICN specific portion 444. The second part is the IMH header 446 which is common to all applications. The third part is the SAI header 448 and the last part is the message payload 450. The ICN header is used by the System Services Subsystem to route messages to tasks. The IMH header is used by the application software to determine the format of the application header.

Detailed Description Paragraph Right (354):
The SAI header is shown in FIG. 58. The To object identifier 644 identifies the object that originated this message. This field is generally never modified by the receiver of the transaction. The transaction ID 646, in conjunction with the to object identifier, uniquely identifies the instance of a message. It is used to match up application commands and responses, detect tardy responses, and detect lost responses. Applications calling a synchronous SAI send operation provide a unique transaction identifier for each transaction based on the object instance ID and transaction number. The status field 648 is used to indicate the general status of the SAI operation. Detailed status information is part of the message payload. The content of the message payload 450 is specific based on the SAI message type.

Detailed Description Paragraph Right (355):
The SAI provides synchronous blocking, synchronous non-blocking and asynchronous protocols for communication. The synchronous blocking interface allows subsystems to obtain indirect access to entities contained in other subsystems. A message is sent to the subsystem managing the entity and the caller's task is blocked until a response for the message is received or a time out occurs. The asynchronous interface allows subsystems to obtain indirect access to entities contained in other subsystems. A message is sent to the subsystem managing the entity, and control is returned to the caller's task. The synchronous non-blocking interface allows a subsystem direct access to entities contained in other subsystems via a function call. This interface is provided for time critical operations which cannot afford the overhead of messaging to access entities in other subsystems. This method of accessing entities is generally limited to a few special cases.

Detailed Description Paragraph Right (356):
The SAI does not manage errors, but rather reports them. Errors that occur when constructing a message use an error envelope to throw an exception. Examples of such errors are (1) errors encountered on ICN call to allocate a ICN buffer; and (2) constructor argument out of range. Methods which initialize message attributes return an error status code if the value of a particular attribute is determined to be out of range. The object communication component returns an error status code if errors are encountered. Examples of possible errors are: (1) errors encountered on ICN calls; (2) messages that are not properly addressed; and (3) time out expires on a synchronous send operation. The object communication component uses an error envelope to throw an exception if it receives messages with a class other than SAI.

Errors encountered during message construction use an error envelope to report the error. The object communication component uses an error envelope to report messages received that are not SAI class messages.

Detailed Description Paragraph Right (357):
No special controls are needed to debug the SAI internals. The SAI can to capture information which can aid in debugging system-wide problems. The SAI supports internal debugging designed for use during product development and testing, and product debugging designed for use during the deployment of the product. Internal debug consists of logging to RAM or an output debug port. This debugging is designed for use during the unit, component, subcomponent, and system test phases. This facility is disabled during the product test phase and is not available in the deployed product. Since the SAI does not have its own task context, logging is accomplished by sending a message to a logging task. At a minimum, the debut control bit mask for the SAI comprises (1) Debug Control to enable/disable RAM logging and External I/O logging; and (2) Object Class to filter SAI transactions on the basis of the target object class. Logging is generally only available in the object communication component of the SAI.

Detailed Description Paragraph Right (358):
To conserve RAM resources, identifiers are logged in numeric format. These identifiers are translated to ASCII text when the debug log information is presented at an external interface. The SAI Debug Entry Format is shown in FIG. 59. The debug entry comprises (1) To ACID 650--destination ACID of the message; (2) To SAOID 652--destination SAOID of the message; (3) From SAOID 654--object source of the message; (4) Flags 656--flags of the IMH header; (5) Sequence No. 658--Application specified sequence number; (6) Message ID 660--Numeric message identification code; (7) SAI Status 662--Status field of the SAI header; and (8) Time 664--Date and time stamp of the operation. External debugging is supplied for the collection of debug information when the product is deployed in a customer environment. Trace statements are logged as strings for all SAI transactions. These strings comprise: (1) SAI--indicates SAI transaction; (2) Message ID--ASCII representation of numeric message ID code; and (3) To SAOID--ASCII representation of target SAOID.

CLAIMS:

1. Network management apparatus for interfacing a network element in an access system with at least two operations systems employed in an open system interconnection or OSI architecture, and a non-OSI architecture, respectively, the network management apparatus comprising:

a first interface for interfacing said network element to said OSI operations system;

a second interface for interfacing said network element to said non-OSI operations system;

a control device coupled to the said first interface and said second interface for receiving OSI and non-OSI operations system commands from said first interface and said second interface, respectively; and

a memory device coupled to said control device for storing a database of managed OSI-based object instances organized into object classes, and at least one table of data comprising an access identifier, at least one of said object classes and an object instance identifier corresponding to a non-OSI command;

wherein said control device being programmable to process said OSI commands using said database and said non-OSI operations system commands using said table to create messages for requesting services on said object instances.

3. A method of generically mapping a Transaction Language 1 or TL1 command into a Common Management Information Service or CMIS-type service, comprising the steps of:

generating TL1 proxy objects corresponding to at least one of a plurality of

CMIS-type services requested in said TL1 command;

generating a service message for delivery to a memory device which stores object instances organized into classes;

retrieving object classes stored in said memory device to which said service message is directed and saving said classes until corresponding ones of said plurality of services are complete;

performing said generating TL1 proxy objects step, said generating a service message step and said retrieving step to complete each of said plurality of services and correspondingly modify respective ones of said saved classes from a first state in said memory device;

generating a success message after said plurality of services are all complete; and

generating a failure message if any one of said plurality of services fails to be completed, and restoring said object class to which said failed service was directed back to said first state.

5. A control system for a network element apparatus in an access system, the network element comprising common equipment for supporting a plurality of different subscriber systems, the common equipment having a number of circuit packs for performing functions common to all of the subscriber systems, and the subscriber systems each also having at least one circuit pack for performing functions associated with that subscriber system, the control system comprising:

at least one processor in each of said plurality of subscriber systems, said plurality of subscriber systems being connected to at least one of said common equipment circuit packs and operable in accordance with at least one of a plurality of application subsystems;

at least one processor in said common equipment for controlling the operation of the circuit packs therein and for communicating with said at least one of subscriber system processors; and

at least one memory device connected to and accessed by said common equipment processor, said memory device comprising:

a collection of objects, subsets of said collection of objects corresponding to respective ones of said application subsystems; and

at least one subagent stored therein and used by said common equipment processor for supporting each of said application subsystems;

said common equipment processor being programmed to operate a common subagent interface along which each of said subagents can generate and transmit messages to other ones of said subagents and receive messages therefrom.

6. A network element control system as claimed in claim 5, wherein said subagents are configured to send messages to at least one object from said collection of objects that is associated with a destination application subsystem, said common equipment processor being operable to route said messages to said destination application subsystems transparently with respect to said originating subagents using said object such that the organization of said collection of objects and said subsystems is concealed from said originating subagents.

10. A network element control system as claimed in claim 9, wherein said collection of objects comprises a session object class defining messages to validate commands from one of said external management systems and to inhibit and allow messages during a session initiated by a user of said external management system.

11. A network element control system as claimed in claim 6, wherein at least one of said plurality of application subsystems is a turnup subsystem for performing at least one of a group of functions consisting of coordinating the start-up functions

of said circuit packs, downloading software to said circuit packs and verifying the correct software load is present on said circuit packs, and said collection of objects comprises a turnup subagent object class defining messages to control the advancement of said turnup subsystem through a state model.

14. A network element control system as claimed in claim 6, wherein at least one of said plurality of application subsystems is a call processing subsystem for controlling the assignment of channels associated with said network element among said subscriber systems, and said collection of objects comprises a call processing subagent object class defining messages to control the advancement of said call processing subsystem through a state model.

15. A network element control system as claimed in claim 6, wherein said subagents are each configured to place at least one of a plurality of subagent object instance identifiers in each of said messages to identify an object from said collection of objects, each of said plurality of subagent object instance identifiers corresponding to one of said objects and one of said application subsystems which operates on said object.

16. A method of controlling a network element apparatus in an access system comprising a plurality of different subscriber systems, each subscriber system having at least one circuit pack and a software subsystem to manage the circuit pack, and common equipment for supporting the subscriber systems having a number of circuit packs and software subsystems for performing functions common to all of the subscriber systems, the method comprising the steps of:

storing a collection of objects in a memory device connected to said common equipment, subsets of said collection of objects corresponding to respective ones of said software subsystems;

generating and storing an object identifier uniquely mapping one of said collection of objects to one of said software subsystems; and

transmitting a message which comprises said object identifier to one of said collection of objects from one of said subsystems.

17. A method for controlling a network element apparatus as claimed in claim 16, wherein said transmitting step comprises the steps of each of said subsystems transmitting messages comprising one of a plurality of said object identifiers to other said subsystems along a common interface provided by a central processor in said common equipment, and further comprising the steps of:

receiving messages comprising one of said plurality of object identifiers along said common interface; and

determining from said object identifier one of said subsystems and a transaction to be performed thereby, said object identifier in each of said messages being useful to conceal the objects and subsystems of said receiving subsystems from said transmitting subsystems.

18. A control system for a network element apparatus in an access system comprising a plurality of different subscriber systems, each subscriber system having at least one circuit pack and a software subsystem to manage the circuit pack, and common equipment for supporting the subscriber systems and having a number of circuit packs and software subsystems for performing functions common to all of the subscriber systems, the control system comprising:

a communications network associated with said common equipment and connected to a plurality of said circuit packs associated with said common equipment and said plurality of subscriber systems to transport messages therebetween, said circuits packs and said software subsystems being characterized as applications, said messages comprising circuit pack identifiers corresponding to source and destination circuit packs, respectively, and application class identifiers specifying at least one of a plurality of tasks that can be performed by said source and destination circuit packs, and a message payload, said communications network being configured to transport messages between applications using application class identifiers while concealing physical board addresses from said source and destination circuit packs.

19. A control system as claimed in claim 18, wherein said messages each further comprise an inter-process message header, said inter-process message header comprising a message class that is defined for each of said subsystems and for messages that are not part of one of said subsystems.

20. A control system as claimed in claim 19, wherein said message class specifies the manner in which said message payload is to be de-coupled and delivered via said communication network, said communication network being operable to transport messages between said subsystems when said subsystems employ different protocols.

21. A control system as claimed in claim 19, further comprising a memory device accessible by said common equipment for storing a collection of objects, said communications network being configured to transport messages between said software subsystems, said software subsystems being operable to communicate with other ones of said software subsystems by sending messages to selected objects in said collection of objects, said messages each comprising an object identifier which uniquely maps an object in said collection of objects to one of said subsystems.

22. A control system as claimed in claim 21, wherein said message class is operable to decouple different message formats used between said circuit packs and between subsystems.

24. A control system as claimed in claim 19, wherein communications network is configured to support a plurality of delivery services for the transport of said messages, said inter-process message header comprising data indicating which of said delivery services is to be used when transporting said message or a response thereto.

25. A control system as claimed in claim 24, wherein said plurality of delivery services comprises services selected from a group of services consisting of: single and multiple attempts to deliver a message, transmission service, error recovery service, post back signal generation circuit upon delivery failure, and route diversity when delivery by a first route fails.

26. A method of controlling a network element apparatus in an access system comprising a plurality of different subscriber systems, each subscriber system having at least one circuit pack and a software subsystem to manage the circuit pack, and common equipment for supporting the subscriber systems and having a number of circuit packs and software subsystems for performing functions common to all of the subscriber systems, the method comprising the steps of:

defining a plurality of applications corresponding to different ones of said circuit packs and said software subsystems, said applications each being characterized by a plurality of application tasks that are performed by respective ones of said circuit packs and said software subsystems;

generating a message from one of said applications for delivery to another one of said applications, said message comprising a communications network header and an inter-process message header, said communications network header comprising a circuit pack identifier and an application class identifier, wherein said application class identifiers are assigned to corresponding tasks that can be performed by a circuit pack; and

de-coupling said message to ascertain said circuit pack and said application task on said circuit pack to which said message is directed.

27. A method as claimed in claim 26, wherein said message further comprises an application header, and said inter-process message header comprises a message class, said de-coupling step further comprising the step of determining from said message class which of a number of formats is employed for said application header.

28. A method as claimed in claim 27, wherein said application header specifies one

of a group of delivery services consisting of: single and multiple attempts to deliver a message, transmission service, error recovery service, post back signal generation circuit upon delivery failure, and route diversity when delivery by a first route fails.

29. A method as claimed in claim 26, further comprising the steps of:

generating a collection of objects representing said application tasks that can be performed by each subsystem;

generating a message comprising an object identifier for delivery between at least two of said software subsystems;

allocating a buffer associated with said communications network for transmitting said message;

determining from said object identifier which of said subsystems and the target application class identifier to send said message to via said communications network;

transmitting said message to said target application class identifiers via the communications network;

determining message type and accessing message payload in said inter-process message header using a message class library associated with said applications.

30. A method for mapping a command received from an external manager by a network element apparatus in an access system into a message having a format which can be processed by the network element apparatus, comprising the steps of:

generating at least one table of data in a memory device within the network element apparatus relating the command and parameters associated with the command to a message identifier;

converting said message identifier to a binary value within a predetermined range of values using a look-up table stored in said memory device, said look-up table relating said message identifier to one of said range of values, to a service provided on a communications network within said network element apparatus, and to at least one of a plurality of object classes which relate a plurality of objects stored in said memory device to each other;

generating an object class identifier using an address identifier within said command in combination with said object class;

converting said parameters in the command to message attributes; and

generating a message from said object identifier, said message identifier and said attributes to invoke a service on at least one of said objects.